



Draft
Alya
Performance Assessment Report

Document Information

Reference Number	PP_AR_??
Author	Michael Wagner (BSC)
Contributor(s)	
Date	08.02.2018



Notices:

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "676553".

© 2018 POP Consortium Partners. All rights reserved.

Content

1. Background	2
2. Application Structure	3
3. FOA (Focus of Analysis)	3
4. Scalability.....	4
5. Efficiency.....	5
6. Load Balance	6
7. Computing Performance	6
8. Communications	8
9. Threading.....	8
10. Accelerators	8
11. I/O	8
12. Summary and Suggestions	9

1. Background

Applicants Name:	Herbert Owen (BSC)
Application Name:	Alya
Programming Language:	Fortran
Programming Model:	MPI + OpenMP
Source Code Available:	Yes (not accessed)
Input data:	Test problem
Performance study:	Initial audit to identify areas for improvement

The application was monitored by the applicant on Mare Nostrum 4 at BSC, a system based on Intel Xeon Platinum 8160 with 48 cores per node (2x 24 cores per chip). The applicant recorded traces using 1, 2, 4, 8, and 16 nodes, i.e. 48, 96, 192, 384, and 768 cores, respectively. All measurements are recorded in MPI-only mode. The traces are used to study the application behaviour in a strong scaling setup. All traces were collected with Extrae 3.5.2 using detailed trace mode with no sampling and recording of hardware counters in two sets changing every 0.5 seconds.

2. Application Structure

Figure 1 depicts the timeline of the execution using 48 cores, i.e. one node. The colour gradient from green to blue represents the duration of the compute phases. After a large initialization phase (about first half, marked in orange) the code executes 50 iterations. Thereby, all iterations show nearly similar behaviour.

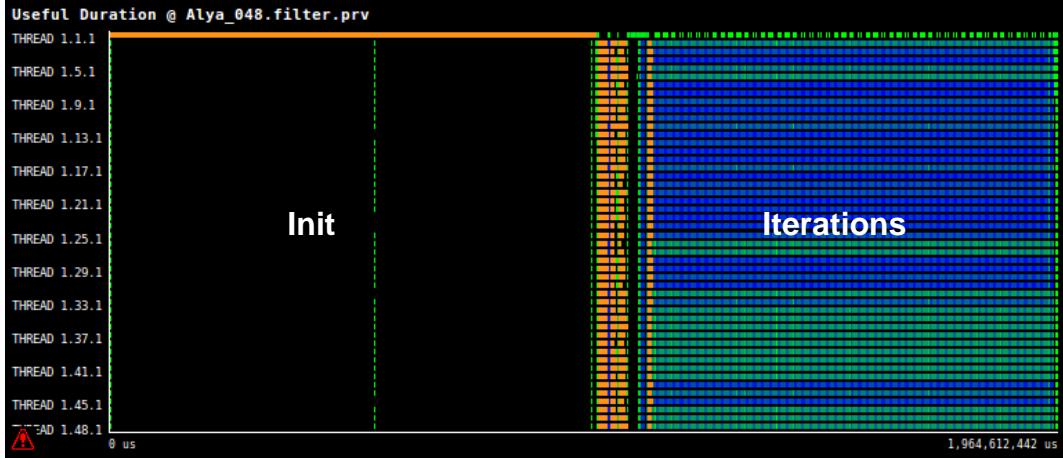


Figure 1. Application structure in a timeline view using 48 cores.

3. FOA (Focus of Analysis)

The iterations show only marginal deviations along time. To exclude variable effects from the initialization phase and towards finalization we selected the fifth iteration from the execution as focus of analysis (FOA). Figure 2 depicts the distribution of computation phases (left) and the parallel behaviour (right) of the FOA.

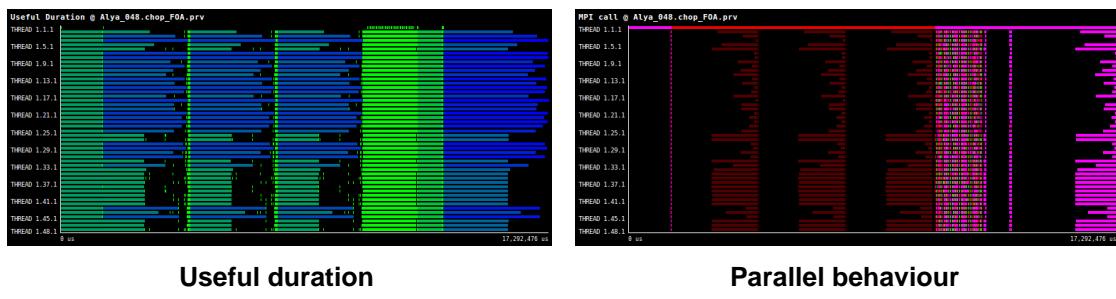


Figure 2. Focus of Analysis (FOA) using 48 cores (1 node).

Each iteration consists of seven main compute phases, whereas in each the first process is not participating in the computation. The first phase is nearly balanced and terminated by a call to *MPI_Allreduce* (pink, including the first process). Phases 2, 3, and 4 are nearly identical and rather imbalanced. After each phase the active processes are synchronized by calls to *MPI_Sendrecv* (dark red) and finally synchronized with the first process by an *MPI_BARRIER* (red). After that follow two smaller phases synchronized with *MPI_Allreduce*. Finally a larger imbalanced phase again synchronized with *MPI_Allreduce*.

4. Scalability

Figure 3 highlights the scalability of the FOA. It shows the execution structure of the compute phases of the FOA on the left side; whereas the time is normalized to 100% of the FOA duration. It highlights that all compute phases scale relatively similar.

The right side depicts the speed up of the FOA in comparison to the smallest run with 48 cores. In a perfectly linear strong scaling execution we expect that each time the number of cores doubles, the total execution time of the FOA reduces by half (red line on the Speedup chart at the right side of Figure 3). The overall scaling of the FOA is very good with a speedup of 14.1 out of 16 (88%).

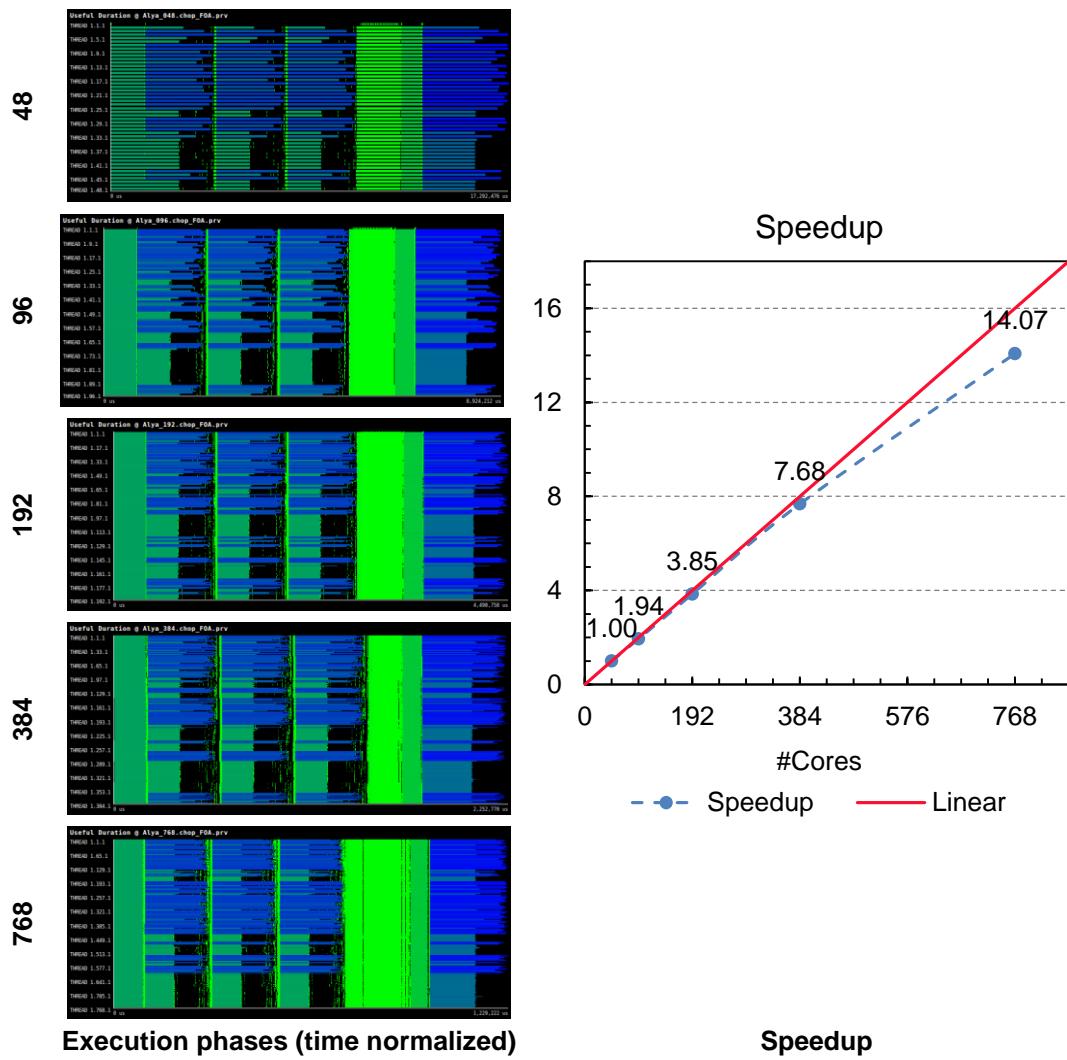


Figure 3. Scalability of FOA. Timeline of execution phases and speedup chart.

5. Efficiency

Table 1 and Table 2 show metrics for fundamental factors and efficiencies from the FOA of the executions using 48 to 768 cores. Values are in percentages with higher values being better.

The observed global efficiency of the application decreases from 79.0% with 48 cores to 69.5% with 768 cores. The decreasing global efficiency is mainly caused by a decreasing communication efficiency (i.e. relatively more time is spent in communication) and decreasing load balance that is already rather low for the smallest measurement causing a rather low global efficiency to begin.

The low load balance of the application is discussed in more detail in Section 6. The communication efficiency, although slightly decreasing, achieves a very good scalability and is detailed in Section 8. The scalability of the computation is very good, too, with a slight anomaly at 384 cores (see Section 7).

	48	96	192	384	768
Parallel Efficiency	79.06%	77.15%	76.52%	73.08%	71.92%
↳ Load Balance	79.39%	78.16%	77.83%	74.97%	76.25%
↳ Comm. Efficiency	99.58%	98.71%	98.32%	97.49%	94.31%
↳ Serialization	99.67%	98.91%	98.73%	n.a.**	97.09%
↳ Transfer	99.92%	99.79%	99.59%	n.a.**	97.14%
Computation Scalability*	100.00%	99.29%	99.45%	103.79%	96.66%
Global Efficiency	79.06%	76.60%	76.11%	75.86%	69.51%

Table 1. Time efficiencies for the FOA.

	48	96	192	384	768
IPC Scalability*	100.00%	99.59%	100.22%	102.56%	99.80%
Instructions Scalability*	100.00%	99.76%	99.34%	101.40%	97.08%

Table 2. Other efficiencies for the FOA.

* Reference values are based on the measurement with 48 cores

** For the measurements with 384 cores an error in the trace prevented the detailed computation for serialization and transfer efficiency without affecting the remaining values.

6. Load Balance

The observed measurements show generally a rather low load balance that is decreasing with an increasing number of cores. Figure 4 highlights the load balance in a timeline for 192 cores. The upper part shows the distribution of the computation in timeline view, whereas the colour gradient shows the duration of each compute phase from short (green) to long (blue). It highlights that Phase 2-4 and Phase 7 are the main source for the imbalance, while Phases 1, 5, and 6 are rather balanced.

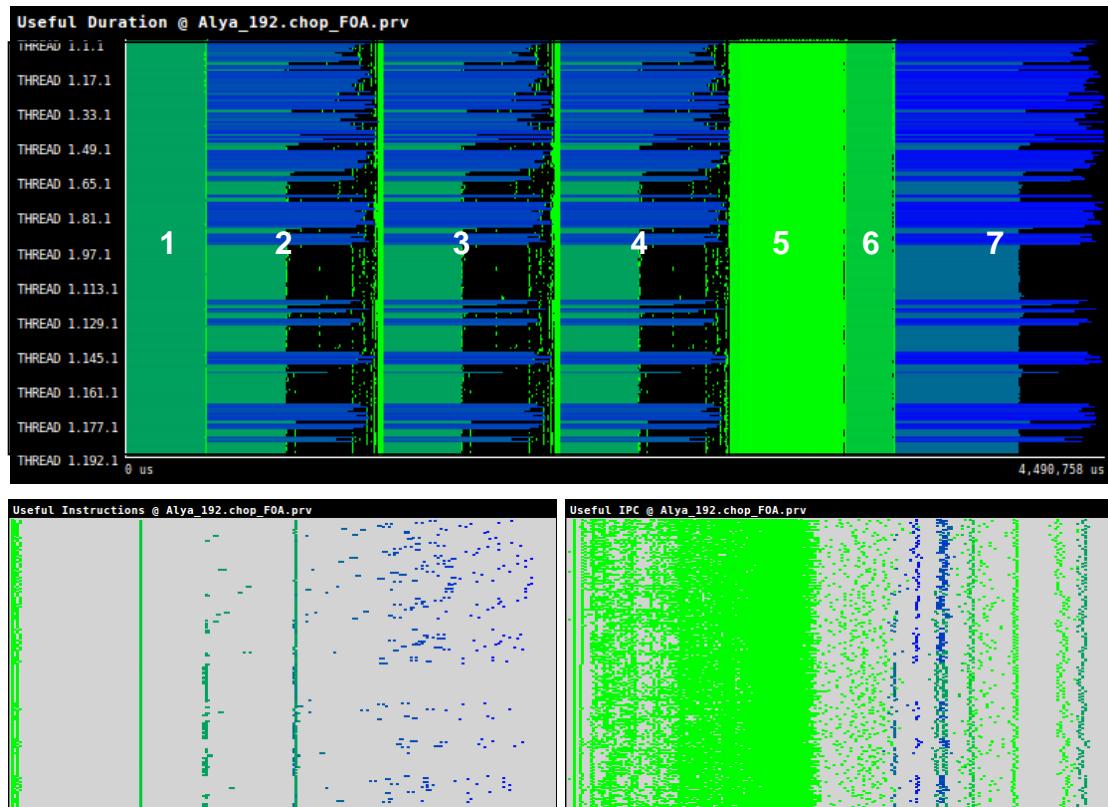


Figure 4. Load balance of the FOA for 192 cores. Top: timeline of computation phases. Bottom: distribution of instructions (left) and IPC (right).

The histograms at the bottom of Figure 4 depict the distribution of instructions (left) and IPC (right) for the above compute phases. The histograms represent for each process on the vertical axis the distribution of compute phases categorized by their number of instructions and IPC (horizontal axis); whereas, the colour of each phase is identical with the timeline on the top. The deviations of compute phases illustrate the grade and location of imbalance; a perfectly balanced phase would form a straight line from top to bottom.

It can be seen, that the imbalance in execution time strongly correlates with the imbalance in the number of instructions (balance is about 69% and 75% for Phases 2-4 and Phase 7, respectively), while the IPC is well balanced (balance is about 97% and 96% for Phases 2-4 and Phase 7, respectively). This means, the origin of the imbalance in time is directly linked to an imbalance distribution of the workload to the processes, i.e. some processes have more than twice the work than others. Due to the highly irregular pattern that is persistent for different scales it is plausible that it is linked to the input data set.

7. Computing Performance

The observed computing performance of the application averages at about 2.0 instructions per cycle (IPC). In general, the application achieves a very good computing performance for this machine.

To further distinguish the individual compute phases we applied clustering, which groups compute phases with similar performance behaviour. Figure 5 shows the compute phases detected by clustering. It includes four six clusters: Cluster 1 (light green) and Cluster 3 (red) correspond to the two variations of the Phases 2-4. Similarly, Cluster 2 (yellow) and Cluster 6 (violet) correspond to the two variations of Phase 7. The Clusters 1 and 5 represent the Phases 1 and 6, respectively. The compute regions of Phase 5 are filtered due to their very short duration.

The right side of Figure 5 depicts the individual compute phases of each cluster based on their number of instructions (vertical axis) and achieved IPC (horizontal axis). The large vertical dispersion of Clusters 1 and 3 and Clusters 2 and 6 underline again the large load imbalance in these two phases the correlates to the distributed workload. It details the pattern of the imbalance: about two thirds of all processes (Cluster 1) have a varying but generally high workload and the remaining one third of the processes (Cluster 3) has a rather equal workload that is about half of Cluster 1. Similar behaviour can be observed for Clusters 2 and 6. However, they have a slight tilt to upwards right, i.e. processes with a high workload have a higher IPC as the ones with lower workload, which conceals the load balance a bit in the resulting execution duration.

In general, all clusters achieve a very good compute performance with Clusters 2 and 6 having the lowest performance with the highest workload and Cluster 4 having the highest performance. When scaling up, the IPC as well as the number of issued instructions scales almost perfectly, i.e. the workload distribution scales very well in the measured range.

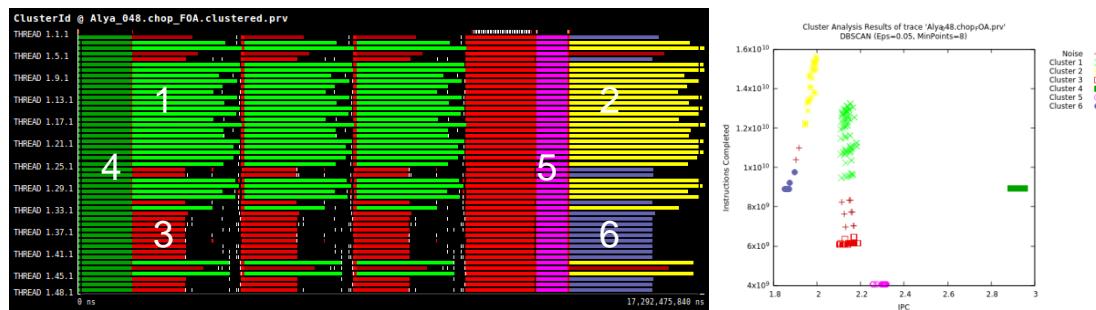


Figure 5. Clustering of compute phases in the FOA with 48 cores. Left: cluster appearance over time. Right: all compute phases clustered with the number of instructions on the vertical axis and the average IPC on the horizontal axis.

8. Communications

The main MPI communication patterns are globally synchronising calls to *MPI_Allreduce* (pink) at the end of Phases 1, 6, and 7 and de-facto synchronising calls to *MPI_SendRecv* (dark red) at the end of Phases 2, 3, and 4. As these calls are synchronising over all processes they catch the previous load imbalance, while introducing only minimal additional delay, i.e. once the last process finishes his computation and enters the communication phase, it is rapidly completed.

The communication efficiency slightly decreases with an increasing number of cores mainly because the total amount of data that is transferred does not scale ideally. Nonetheless, the overall communication efficiency is very good.



Figure 6. MPI communication in the FOA for 48 cores.

9. Threading

This section does not apply for this audit.

10. Accelerators

This section does not apply for this audit.

11. I/O

This section does not apply for this audit.

12. Summary and Suggestions

In this audit we analysed the performance of Alya to identify general areas for improvement. We analysed traces based on 48, 96, 192, 384, and 768 cores. Overall, the application achieves a very good scalability for the given range. The application realizes almost perfect scalability in the computation, i.e. the workload is almost perfectly distributed with increasing core counts. In addition, the application shows a very good communication efficiency (MPI parallelization).

We found the main issue of the application to be load balance, which is already decreased on the smallest run and continues to decrease slightly when increasing the number of cores.

- In the computation phases 2, 3, 4, and 7 the application exhibits a high load imbalance that is directly correlated to the number of executed instructions, i.e. the origin of the imbalance in time is directly linked to an imbalance distribution of the workload to the processes, where some processes have more than twice the work than others.
- The load imbalance is the primary target to improve the overall performance of the application. It would allow a runtime improvement of up to 20% -24% for the given scale.
- Due to the highly irregular pattern of the imbalance it should be evaluated whether the imbalance is directly related to the characteristics of the used input dat.